UNIVERSITY OF PISA

Computer Engineering

Internet of Things

Load Forecasting and Autonomous Shed-Load Control for Smart Buildings

Professors:

Giuseppe Anastasi

Francesca Righetti

Carlo Vallati

Student:

Francesco Taverna

ACADEMIC YEAR 2024/2025

Chapter 1

Introduction

In modern power systems, the integration of intelligent, automated demand-side management is crucial for ensuring efficiency, reliability, and resilience. This project presents an IoT-based solution for **real-time load forecasting and autonomous shed-load control** within a smart building context. The system is designed around resource-constrained IoT nodes capable of performing on-device load prediction and executing local control actions without reliance on continuous cloud connectivity.

The node samples electrical parameters (active power, derived from voltage and current sensors) at fixed intervals and uses a **lightweight machine learning model** to forecast the electrical load. If the predicted load exceeds a configurable threshold, a **relay is triggered** to temporarily disconnect non-critical electrical loads, effectively implementing a localized **demand response** strategy. A correction factor, calculated using real-time sensor data, adjusts the model's prediction dynamically to increase reliability.

This approach is motivated by key challenges outlined in the article *"A Comprehensive Review on Internet of Things Applications in Power Systems"*, which highlights that:

"Some of the most essential challenges in traditional power systems are fault detection, blackouts, load forecasting, energy theft, and transmission losses."

By combining **local intelligence** with lightweight protocols and real-time control, this system provides a practical and scalable example of how IoT can enhance energy efficiency and operational autonomy in smart grid environments.

Chapter 2

Use Case: Load Shedding

The Load Shedding system (Figure 2.1) is designed for preventing grid overloading due to excessive power consumption and for managing energy in an efficient way in a smart building. This can be achieved by turning off less relevant devices in the building when the predicted power is over the threshold.

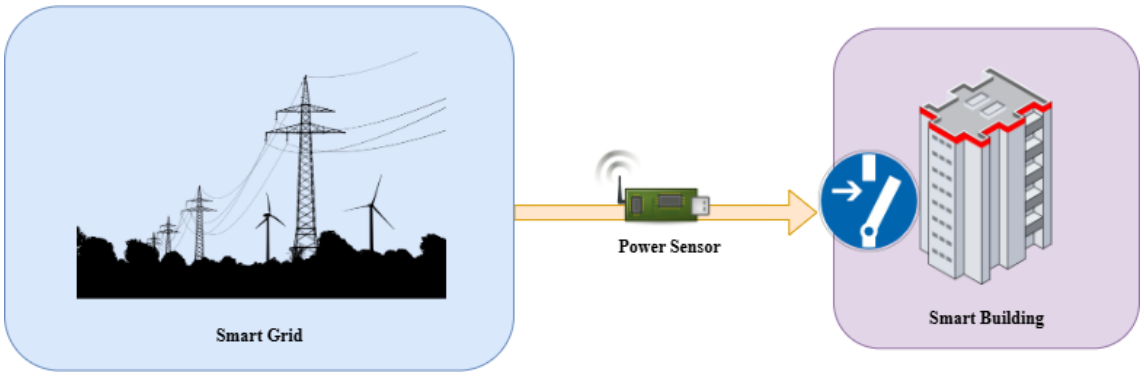

Figure 2.1: A simple scheme of the Load Shedding system

System Overview

The system is composed of two parts:

- **Power Node**: it senses the current power consumed by the building and predicts the incoming load. This data will be sent to a cloud application to let users monitor the current load and to the smart building actuator.
- **Shed Load Relay**: it is the actuator used to turn off smart building non-critical devices and systems when the predicted power consumption is over the configured threshold.

System workflow

All the performed operations are part of the following periodic cycle:

1. The current building load is sampled by the relative sensor.
2. The machine learning model produces a prediction of the future consumption, thanks also to the sample, and sends it to the actuator.
3. If the predicted value is over the threshold, set by the user, the relay will unsupply the less important systems of the building. When the predicted load is back to a normal value, the actuator is turned off.

Chapter 3

System Implementation

The Load Shedding system (Figure 3.1) exploits a simple wireless sensor network (WSN) to track the smart building consumption in order to avoid blackouts due to grid overload. In this chapter it is analyzed how the system works and how it is implemented.

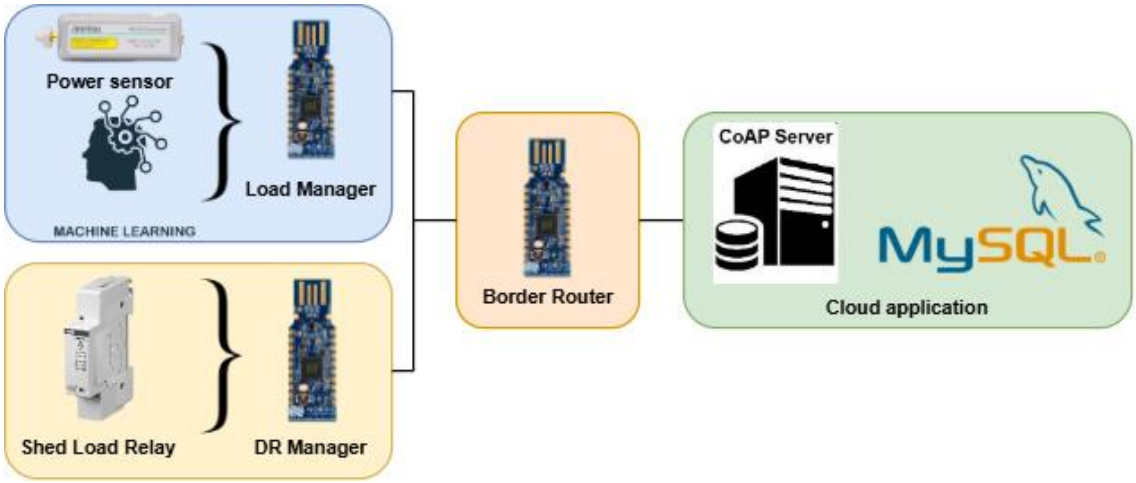

Figure 3.1: Load Shedding system architecture.

Why Coap

The system employs the Constrained Application Protocol (CoAP) instead of MQTT to better support local, autonomous decision-making and time-sensitive control operations. Unlike MQTT, which requires a centralized broker often located in the cloud, CoAP allows for **direct machine-to-machine interaction** between IoT nodes. This is particularly important in the proposed architecture, where the actuation logic (e.g., relay control for load shedding) must be executed **locally** by the sensor node based on the current timestamp and predicted load. By eliminating the dependency on a broker, the system ensures **faster response times** and adheres to real-time constraints, if any, that could be critical for effective demand response. Additionally, CoAP's decentralized nature enhances **resilience**, allowing nodes to continue operating and exchanging data even in case of temporary disruptions at the border router or network gateway.

System Components and Interaction

The Load Manager and the Demand Response Manager shown in the previous image are the components that have the goal to interact with the environment. In addition to them, a third node, called Border Router, is used to allow communication between the WSN and the Internet.

Load Manager

- **Resource Exposed:** */load*, */settings*.
- **Function:** Samples current load of the building and compute the prediction.
- **Observations:** Does not observe any other resource.

This node is essentially a sensor since it does not take any action on the environment, however it has the assignment to establish when a new cycle of operation starts. The operations of the node are divided into the following phases, each one is described by a LED color:

1. **Configuration Phase**: (YELLOW)
   This phase happens only once after the device is turned on. It is composed of:
   (a) **Main Resource Activation:** */load* is activated for permitting other nodes to subscribe to it.
   (b) **Clock request:** the node sends a request to the CoAP Server to retrieve the exact time. This is necessary for the operation of the **machine learning model**.
   (c) **CoAP Server registration:** the node registers to the server, sending its initial settings and information.
   (d) **Secondary Resource Activation:** */settings* is activated for permitting the user application to modify it.
   (e) **Initialize the timer:** a timer is set to start the next phase. The latter will start after *sampling_period* seconds (configuration variable of */settings* resource).
2. **Operation Phase** (GREEN)
   This phase is called every time the timer expires (15 minutes is the sampling period in a real use case), given that this is the operation at time t_i:
   (a) **Load sampling:** thanks to the relative sensor, sample L_i is generated.
   (b) **Load prediction:** the node computes the timestamp t_{i+1} and the load prediction P_{i+1} using the machine learning algorithm (MLA):

   $$P_{i+1} = MLA(t_{i+1}) + CorrectionFactor_i$$

   Where the Correction Factor is defined as follows:

   $$CorrectionFactor_i = (L_i - P_i) * 0.12 + CorrectionFactor_{i-1} * 0.88$$

   A correction factor was implemented to adapt the predicted load to real-time sensor readings and to allow the system to react to unpredicted anomalies or environmental changes, without discarding the learned structure of the prediction model
   (c) **Notify the observers:** notifying the observers of the */load* resource.

3. **Sleep Phase** (RED)
   In this phase the node is waiting for a new event.

Demand Response Manager

- **Resource Exposed:** */status*, */settings*.
- **Function:** Activates the Shed Load Relay if the power consumption is over the threshold.
- **Observations:** */load*.

This node is an actuator that can turn on and off the relay. Indeed, the color of the LED identifies the status of the relay:

- **RED:** the shed load relay is off.
- **BLUE:** the shed load relay is on.

The other association with LED's color is expressed in the following phases:

1. **Configuration Phase** (YELLOW)
   This phase happens only once after the device is turned on. It is composed of:
   (a) **Main Resource Activation:** */status* is activated for permitting other nodes to subscribe to it.
   (b) **Clock request:** the node sends a request to the CoAP Server to retrieve the exact time. Although this is not necessary for the execution of the main operation because the timestamp received by the Load Manager will be used, it was decided to maintain the consistency of the messages, even in the first message sent to the CoAP server.
   (c) **CoAP Server registration:** the node registers to the server, sending its initial settings and information.
   (d) **CoAP Discovery and Subscribing:** the node asks the CoAP server the ip of the Load Manager for subscribing to the */load* resource.
   (e) **Secondary Resource Activation:** */settings* is activated for permitting the user application to modify it.
2. **Operation Phase** (Relay status color)
   This phase starts when a notification from the */load* resource arrives. Initially, the relay is off (RED led). It is composed of:
   (a) **Load Data Reception:** with the notification new data are received, in particular the sampled and predicted load.
   (b) **Managing the Shed Load Relay:** if the predicted load is above the threshold, the relay is turned on (BLUE led on), otherwise it is turned off (or remains off if it was already) (RED led on).
   (c) **Notify the observers:** notifying the observers of the */status* resource.

3. **Manual Phase** (Relay status color)
   This phase is triggered when the button is pressed: the led will turn RED if it was BLUE, BLUE if it was RED. This inversion remains effective until the timer expires, then, at the arrival of the new load, the status could be inverted again according to the update rule. This was done to act quickly.
4. **Sleep Phase** (Relay status color)
   In this phase the node is waiting for a new event.

Border Router

- **Function:** Provides connectivity between WSN and the internet.

The border router facilitates remote access and control of the system, enabling seamless communication between local sensors and cloud-based applications.

CoAP Server

- **Resource Exposed:** */clock*, */discovery*, */registration*.
- **Function:** Manages node registration, insertion of new resource value in the database and current clock sending
- **Observations:** */load*, */status*.

The CoAP Server handles the initial registration of the nodes as they join the network, furthermore, collects and stores load data and relay status data in a MySQL database. This can be achieved by observing each main resource of a new registered node.

User Application

- **Function:** Allows the dynamic control of some parameters of the nodes and its listing.
- **Observations:** Does not observe any other resource.

The user application implements three essential functions: list all registered nodes, modify actuator status and dynamically change some parameters of the nodes, such as the sampling interval and load threshold. In addition, it is also possible to see past samples, predictions and actuator statuses.

Database

- **Function:** Stores all the collected data.

The database is implemented in MySQL.

Chapter 4

Data Encoding

In the Load Shedding system, JSON (JavaScript Object Notation) is used for data encoding: this choice is driven by its simplicity, human readability, and widespread adoption, making it an ideal format for data interchange in IoT environments. Moreover, for the sensor measurements, it adopts the SenML (Sensor Measurement Lists) standard to ensure interoperability and seamless integration with third-party applications.

JSON and SenML

The choice of JSON data encoding is justified by three aspects:

1. **Simplicity:** the JSON format is simple to generate and easy to read for a human operator; this makes the perfect choice for an IoT environment that needs continuous monitoring and debugging.
2. **Lightweight:** in an IoT environment where computational resources are limited, having a lightweight structure increases efficiency and ensures minimum resource utilization.
3. **Globally used:** thanks to the widespread diffusion, using the JSON format facilitates the integration of the system with other third-party applications.

On top of that, the adoption of a more specific format, like the **SenML** standard, is taken for extending further the third point: indeed, the sensors programmed in the Load Shedding system can be utilized in a broader context thanks to this choice.

SenML Structure

SenML format is divided into two types of fields: base and regular.

- **Base fields**
  - **Base Name (bn):** this is a string that is prepended to the names found in the entries.
  - **Base Unit (bu):** a base unit that is assumed for all entries, unless otherwise indicated. If a record does not contain a Unit value, then the Base Unit is used. Otherwise, the value found in the Unit (if any) is used.
- **Regular fields**
  - **Name (n):** name of the sensor or parameter. When appended to the Base Name field, this must result in a globally unique identifier for the resource.
  - **Time (t):** time when the value was recorded.
  - **Unit (u):** unit for a measurement value.
  - **Value (v):** values are represented using basic data types:
    - *v*: floating point numbers.
    - *vd*: integer value.

An example of a Load Manager SenML packet is the following:

```
[
 {"bn":"node","bu":"kW"},
 {"n":"predicted","u":"kW","t":"2025-05-25T12:00Z","v":613417},
 {"n":"sampled","u":"kW","t":"2025-05-25T11:45Z","v":692345}
]
```

Floating point values

Due to the limitation of the NRF52840 microcontrollers, it is not possible to print anywhere any floating point, despite the dongle support the floating-point operations. To resolve this, it is decided to multiply every float number by 10000 and to convert it into an integer right before the data is sent. The cloud application reconverts it to float after the reception. This gives space for only four decimal numbers, but they were considered sufficient for the application. So, for example, 5.3131 kW load will be sent as 53131.

Chapter 5

Machine Learning Model

The Load Shedding system uses a machine learning model for predicting power consumption over the year to turn off less important devices when the load is too high. The chosen model is the Random Forest Regressor known for its robustness and accuracy, and it is implemented within the Load Manager node.

Model Training and Integration

The **Load Manager** node uses real-time timestamps to predict the electrical load of a smart building using 15-minute intervals. Unlike traditional models that work with hourly predictions, this system benefits from a finer temporal resolution, allowing more precise and timely responses. The dataset used for training is the "*Household Electric Power Consumption*" from Kaggle, which contains over 500.000 records of real-time power consumption and other metrics in a household environment. The only features extracted are the *date*, *time* and *global_active_power*. This dataset contains samples taken every minute, so a more fine-grained prediction can be implemented, but, to be more lightweight, a lower frequency sampling has been chosen.
To simplify testing and reduce execution time during development, the system includes a *DEBUG* preprocessor constant. When enabled, it allows the simulation to run with a reduced sampling period while internally maintaining the real 15-minute time advancement logic. This feature can be disabled from the relevant resource files (*settings.c*) to run the system with the realistic time intervals. Despite being trained on real-world data, the model can generalize across different load patterns, and its accuracy is dynamically improved through real-time correction based on sensor readings.

Preprocessing

As stated before, not all the features are necessary to the scope of the application, so in this phase the irrelevant ones are removed. Moreover, all the records that have a 0 or NaN value on the load are removed, in this way the Random Forest Regressor can learn only the significant variation of day consumption.
To reduce the 2M rows dataset, only the first year of measurements is taken. To be more precise, the time interval considered is from 17/12/2006 to 17/12/2007.

Training

The training of the Random Forest Regressor involves four key steps:

1. **Data preparation:** the features (X) and labels (y) are defined.
2. **Dataset splitting:** The data is divided into training and testing sets to ensure the model's performance can be evaluated accurately.

3. **Standardization:** The data is standardized for better performance, the parameters used for the standardization will be integrated in the node's code as a preprocessor variable.
4. **Model training:** The Random Forest Regressor is trained on the training set, learning the relationships between load and the progressing of time.

Testing

After training the Random Forest Regressor, the testing results show a Mean Squared Error (MSE) of 0.59 and a coefficient of determination (R^2) of 0.57. These metrics indicate that the model has an acceptable predictive performance, with the R^2 value suggesting that approximately 57% of the variance in the target variable is explained by the model.

Macro Parameters

These values were calculated by first cleaning and filtering the household power consumption data for a specific one-year period. The dataset was divided into daytime and nightime based on hourly ranges. Key statistics such as minimum and maximum power loads during the day and night were extracted. Then, the average relative increments in power consumption were computed separately for morning and night periods by analyzing the differences between consecutive measurements.

Node Deployment

The model can now be converted through *emlearn* in efficient C code, allowing it to run effectively on the node's hardware.

Chapter 6

MySQL Database Schema

The Load Shedding system uses a MySQL database to store sensor data and actuator statuses. This database is important for maintaining historical records and for guaranteeing a dynamic function of the system. In this chapter, the key schema of the key tables in the database is described.

Table: nodes

In this table are registered all the nodes with the correlated information.

Field	Type	Null	Key
ip	varchar(255)	NO	PRIMARY
name	varchar(255)	NO	
resource	varchar(255)	NO	
settings	varchar(255)	NO	

Table 6.1: Nodes table schema.

As it can be seen from Table 6.1 the fields are:

- **ip:** the ip of the node.
- **name:** the name of the node.
- **resource:** an array of the exposed resources of the node.
- **settings:** a json formatted string that contains all the configurable parameters.

Table: load

In this table, load samples and the relative predictions are collected. In particular given a sample L_i at the time instant i, the respective prediction P_{i-1} will be memorized in the same row to facilitate the comparison of the two values.
As it is possible to see from Table 6.2 the fields are:

- **timestamp:** time instant where the load was sampled.
- **sampled:** the value of the sampled load.
- **predicted:** the value of the predicted load for the same timestamp

Field	Type	Default	Null	Key
timestamp	TIMESTAMP	current_timestamp	NO	PRIMARY
sampled	FLOAT	NULL	YES	
predicted	FLOAT	NULL	YES	

Table 6.2: Load table schema.

Table: relay

In this table the relay status in a certain time instant is collected.

Field	Type	Default	Null	Key
id	INT	AUTO_INCREMENT	NO	PRIMARY
timestamp	TIMESTAMP	current_timestamp	NO	
status	BOOLEAN	false	NO	

Table 6.3: Relay status table schema.

As it is possible to see from Table 6.3 the fields are:

- **timestamp:** time instant where the status was changed.
- **status:** indicates if the relay is on or off:

Chapter 7

Conclusions

The proposed system has been successfully implemented with a functional, lightweight machine learning model capable of forecasting electrical load in real time. Combined with direct sensor measurements and a dynamic correction mechanism, the system enables accurate prediction and timely activation of a shed-load relay. This approach leads to improved energy efficiency and proactive demand response within a smart building context.

While effective in its current form, the system leaves room for expansion and improvement. A future version could incorporate multiple sensors monitoring environmental or operational parameters such as occupancy, temperature, or energy pricing. These additions would enable the deployment of more sophisticated prediction models and more intelligent load management strategies. It would also be feasible to integrate energy storage components, optimizing battery usage by aligning charge and discharge cycles with forecasted demand trends.

In conclusion, this system offers a robust, scalable, and autonomous solution for demand-side energy management in smart grids, particularly suited for buildings, campuses, or microgrid environments. Its design lays a strong foundation for future enhancements aimed at supporting the evolving needs of sustainable and intelligent energy systems.